Yonghoon Choi

Design

**Overview:**

I am building a banking application that processes transactions(open account, withdraw funds, deposit funds, transfer funds, ask for the transactional history to be printed); reads a string of transactions from a file into an in-memory queue. The program will queue and process the transactions in order. When the queue has been finished the program will print out all open accounts and balance in those accounts.

There are going to be 8 different funds of account. (0: Money Market, 1: Prime Money Market, 2: Long-Term Bond, 3: Short-Term Bond, 4: 500 Index Fund, 5: Capital Value Fund, 6: Growth Equity Fund, 7: Growth Index Fund)

There are going to be five types of transactions that are identified by a first character.

**O**: Open a client account and instantiate eight funds with 0 assets each

**D**: Deposit assets into a fund

**W**: Withdraw assets from a fund

**T**: Transfer assets between funds (can be funds owned by a single client or transfers between clients)

**A**: Display the history of all transactions for a client account F: Display the history for all transactions of a given fund

Examples of commands transaction:

(Transaction type, Account id, Fund id, Amount, Transferred from, Destination account)

**D 1234 1 100** - Deposit $100 into the prime money market account of client ID 1234

**W 1234 0 500** - Withdraw $500 from the money market of client ID 1234.

**T 1234 0 1234 1 1000** - Transfer $1000 from client 1234's money market to the prime money market.

**T 1234 0 5678 0 1000** - Transfer $1000 from 1234's money market to 5678's money market.

**A 1234** - Display the history of all transactions of all accounts for client 1234.

**F 1234 4** - Display the history for all transactions on the 500 Index Fund for client 1234

**O 6537** - Bowden Charles Open an account for client Charles Bowden. Use account id 6537.

The program will deal with semantic errors like bad account numbers, or too much amount to withdraw or transfer. However, the money market funds accounts are linked and the bond accounts are linked which means if one linked account does not have enough funds to withdrawal the rest fund will be covered from the linked account if there is enough fund. Appropriate error messages will be printed.

Data Structures:

The STL queue will be used to read the transactions. The accounts will be stored in the binary search tree.

**Class diagrams:**

Diagram

Description automatically generated

**Explanation of how classes:**

JollyBanker class handle the queues of transactions by reading transactions and exit transaction. The class creates an object like ‘bank’ to store clients, clients' accounts, and amounts, and it also initiates the display action in phase 3 that prints out each client account. Transaction class handles different types of transactions (Open, Deposit, Withdraw, Transfer, Display all, Display certain fund). The class will handle the comment from the queue passed in from the Jolly Banker. Account class holds different types of funds’ data that is going to be used for all the transactions and binary search tree. Account class also handles linked funds’ matters. Different accounts for each client will be created by interacting between Account class, Transaction class, and BSTree class, and the data will be handled and saved in Account class. BSTree class handle node actions; creating, inserting, comparing, and deleting. The class also interacts with Account class for binary searching algorithms and checking for the correct amount of each account required for each transaction. Transaction errors checking and printing errors will be handled in Transaction class with interacting BSTree class and Account class for phase 2. Printing from phase two for A and F transactions for phase 2 will be also handled in Transaction class. Phase 3 printing that is initiated in the JollyBanker class will be handled in the Account class at the end by interacting BSTree class.

**Program flow:**

Program5.cpp make an object of JollyBanker class like ‘bank’, transactions are saved and read in the list of queues of Transaction vector of the list in JollyBanker class. Each line of a transaction is excused by Transaction class by saving and handling each clients’ accounts in Account class in Node and binary search tree form with the interacting BSTree class. Printing errors(insufficient funds) and outputs(for A and F transaction) in phase 2 will be executed by the Transaction class if there is any. After phase 2 is over (all transactions are executed) phase 3 (printing out every clients’ accounts with amounts will be handled in Account class which is initiated in JollyBanker class.

**.h files and Important Function Signatures:**

Text

Description automatically generatedText

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated A screenshot of a computer

Description automatically generated with medium confidence